# Python Coding Task Aarthi P

Time: 30 Minutes 20/06/2025

Level: Intermediate

## Q1. Understanding Access Specifiers

Create a class `Student` with the following properties:  
Class Requirements:  
1. `name` → Public attribute   
2. `\_roll\_number` → Protected attribute   
3. `\_\_marks` → Private attribute

Implement the following methods:  
- Constructor to initialize all attributes.  
- `display\_details()` → Public method to display all attribute values.  
- `\_update\_roll\_number(new\_roll)` → Protected method to update roll number.  
- `\_\_update\_marks(new\_marks)` → Private method to update marks.  
- `access\_private\_method(new\_marks)` → Public method that uses the private method `\_\_update\_marks`.

## Q2. Demonstrate Access

In the main section:  
- Create an object of the `Student` class.  
- Modify and print the `name` directly.  
- Modify and print the `\_roll\_number` directly.  
- Try accessing `\_\_marks` directly and observe the result.

## Q3. Inheritance and Access Control

Create a subclass `Topper` that inherits from `Student` and includes:  
- A method `try\_access()` that attempts to access `\_roll\_number` and `\_\_marks` from the subclass.  
- Show what works and what doesn't.

## Q4. Use of Name Mangling

Demonstrate how to access the private attribute `\_\_marks` using name mangling technique from outside the class.

class Student:

def \_\_init\_\_(self, name, roll\_number, marks):

self.name = name

self.\_roll\_number = roll\_number

self.\_\_marks = marks

def display\_details(self):

print("Name of student : ", self.name)

print("Roll Number : ", self.\_roll\_number)

print("Marks Obtained : ", self.\_\_marks)

def \_update\_roll\_number(self, new\_roll):

self.\_roll\_number = new\_roll

print("The roll number of student ", self.name, "has been updated to ", self.\_roll\_number)

def \_\_update\_marks(self, new\_marks):

self.\_\_marks = new\_marks

print("Marks of the student ", self.name, "has been updated to ", self.\_\_marks)

def access\_private\_method(self, new\_marks):

self.\_\_update\_marks(new\_marks)

def getmarks(self):

return self.\_\_marks

class Topper(Student):

def \_\_init\_\_(self, name, roll\_number, marks):

super().\_\_init\_\_(name, roll\_number, marks)

def try\_access(self):

print("Topper's Roll Number ", self.\_roll\_number)

#print("Topper's Marks ",self.\_\_marks)

print("Topper's Marks ",Student.getmarks(self))

print("obtaining marks using mangling ", t1.\_Student\_\_marks)

t1=Topper("Aarthi", 1, 80)

print(t1.name)

print(t1.\_roll\_number)

# print(t1.\_\_marks)

print(t1.getmarks())

t1.try\_access()

## Q5. Reflection

Answer the following short questions:  
1. Why can’t private members be accessed directly?

**Private members (like \_\_marks) are meant to **hide internal details** of a class. Python restricts direct access to help **protect the data** from being changed or misused accidentally.**

2. What is the purpose of using protected members in class design?

**Protected members (like \_roll\_number) are a **gentle warning** to developers: "You can use this, but be careful!" They are meant to be **used within the class and its subclasses**, not outside.**

3. How does name mangling help with private members in Python?

**Name mangling changes the name of private variables internally (e.g., \_\_marks becomes \_ClassName\_\_marks) to make them **harder to access from outside**. It’s not true hiding, but it **adds a layer of safety**.**